Als Computer noch erklärbar waren

Niklaus Wirth

Vortrag zum 40. Geburtstag der Informatik an der ETH am 3. Okt. 2008

1. Einleitung

Das Jahr 1968 ist in die Geschichte eingegangen als Jahr der sozialen Unrast, des Umbruchs. Es war gekennzeichnet durch die Ablehnung von Autorität, die Absage an die Forderung nach Leistung, durch die Befreiung von verkrusteten gesellschaftlichen Konventionen, und durch die Verbreitung der Beliebigkeit. Die Auswirkungen dieses Umbruchs reichen bis in die Gegenwart. Daneben gab es aber auch kleinere, wenig beachtete Ereignisse, an die man sich jedoch heute noch aus guten Gründen erinnert.

Dazu gehört die Einführung der Informatik an der ETH in Zürich. Diese manifestierte sich vor genau 40 Jahren, am 1. Oktober 1968, durch die Gründung der *Fachgruppe Computerwissenschaften* innerhalb der Abteilung für Mathematik und Physik. Davon will ich hier berichten. Es soll auch die Rede sein von deren weiteren Entwicklung bis hin zum selbständigen Departement.

Im allgemeineren Kontext will ich danach die Entwicklung der Computer selbst Revue passieren lassen. Es ist dies eine interessante, von Nichtfachleuten wenig beachtete Geschichte. Am Schluss will ich versuchen zusammenzufassen, was wir daraus lernen könnten.

2. Die ETH führt die Computer-Wissenschaften ein

Der Umgang mit Rechnern hatte an der ETH schon 1968 eine längere Vorgeschichte. Bereits 1950 wurde die letzte Zuse Z4 in Betrieb gesetzt, ein Rechner mit mechanischen Relais und einem Speicher von 64 Zahlen. In den Jahren 1954-58 wurde eine eigene Maschine gebaut. Dazu wurden Elektronen-Röhren verwendet, sowie eine Magnettrommel als Speicher. Diese ERMETH diente bis ins Jahr 1964, als das Rechenzentrum gegründet und mit einem Rechner CDC-1604 ausgestattet wurde. 1970 folgte dann das Grossrechnerpaar CDC 6500/6400. Dieses diente damals noch exklusiv dem *Rechnen* an sich. Die “Kunden” waren Physiker, Chemiker, und Ingenieure.

Das Ereignis von 1968 war insofern bemerkenswert, als die Computer-Technik jetzt als eigene akademische Disziplin eingeführt wurde, die sowohl in Vorlesungen gelehrt als auch in der Forschung untersucht werden würde. In Anlehnung an das amerikanische *Computer Science* hiess das neue Team *Fachgruppe für Computerwissenschaften.* Es bestand aus den Professoren H. Rutishauser und P. Läuchli aus der numerischen Mathematik, sowie mir aus dem Gebiet der Computer-Systeme und der Programmiersprachen. Dazu kamen ein halbes Dutzend Assistenten. Kollege Rutishauser verstarb leider bereits nach zwei Jahren. 1972 stiess Prof. C.A. Zehnder zu uns, und 1974 Prof. J. Nievergelt, womit sich unsere Kompetenzen auf die Datenbanken und die Algorithmik ausdehnten. Zu dieser Zeit wurde die Fachgruppe in *Institut für Informatik* umbenannt.

Schon damals schien uns die Trennung der akademischen Seite vom Dienstleistungsbetrieb des Rechenzentrums wohlbegründet und notwendig. Dort ging es um die Befriedigung von Bedürfnissen der Computer-Anwender, bei uns um die Kenntnisse über die Struktur, die Funktionsweise, und den Bau von Computersystemen selbst. Wir verstanden unser Fach als *technisches* Ingenieur-Fach von zukunftsträchtiger Bedeutung.

Natürlich war unser Sinnen auf die Einführung eines eigenen Studiengangs gerichtet. Bereits 1970 unterbreiteten Zehnder und ich einen konkreten Vorschlag. Nach dessen Ablehnung unternahmen wir 1972 einen zweiten Versuch. Da offensichtlich grössere Unterfangen unerwünscht waren, versuchten wir, der Abteilung für Mathematik und Physik als drittes Standbein die Informatik schmackhaft zu machen. Doch auch mit diesem ökonomischen

Ansinnen war uns kein Erfolg beschieden. Erst nach weiteren neun Jahren wurde 1981 auf sanften Druck aus Bern hin eine neue Abteilung mit eigenem Studiengang gegründet, die erste seit 45 Jahren.

Zu dieser Zeit begann an der ETH eine Periode der Umstrukturierung. Abteilungen (an der ETH damals synonym mit Diplomstudium) sollten Departementen (organisatorischen Einheiten) weichen. Dank Führungsstärke und gutem Willen dauerte diese Umstrukturierung nur 10 Jahre. Eines der allerersten neuen Departemente war die Informatik, und erster Vorsteher war Kollege C.A.Zehnder. Damit begann die eigentliche Wachstumsphase; heute zählt das Departement etwa zwei Dutzend Professoren, 200 Mitarbeiter, und anstatt einem einzigen Hunderte von Rechnern.

3. Computerwissenschaft, eine neue Disziplin?

Weshalb aber entstand eine neue, eigene Disziplin? War das Computerwissen bei der Elektrotechnik und der Mathematik nicht bereits gut untergebracht? Der Computerbau bei der Elektrotechnik, die Computer-Anwendung bei der Mathematik? Aus heutiger Sicht eine fast lächerliche Frage, denn es gibt kaum mehr ein Gebiet, das sich nicht ebenfalls als Computer-Anwender versteht, insbesondere die Elektrotechnik.

Den eigentlichen Anstoss für eine eigene Disziplin gab eine neue Sparte, die weder in die Mathematik noch in die Elektrotechnik richtig passen wollte: die Computer-Programmierung. Dies wurde offensichtlich durch die Entstehung von Programmiersprachen und Systemen, die Programmtexte in Computer-Code übersetzen (Compiler). Eine Programmiersprache – keine eigentliche Sprache, sondern ein Formalismus – stellt eine abstrakte Maschine dar, die einerseits durch mathematisch exakte Regeln definiert ist, und anderseits durch elektronische Bauteile realisiert wird. Abstraktion entpuppt sich als das zentrale Werkzeug der Informatik. Sie ist das einzige Mittel, um die stetig wachsende Komplexität der Computer und der Programme zu beherrschen, um den Programmierer vor der Flut der technischen Einzelheiten zu schützen, damit er sich auf das für seine Anwendung Wesentliche konzentrieren kann.

Etwas überspitzt darf man also behaupten, die Abstraktion, die Programmiersprache, verdecke die Sicht auf die Realität. Dies ist einerseits erwünscht, sogar notwendig, anderseits unerwünscht, nämlich dann, wenn die konzeptuelle Distanz von der Sprache – der abstrakten Maschine – zum Computer – der realen Machine, gross und daher der Uebersetzungsprozess komplex und undurchsichtig ist. Dann könnten nämlich Kenntnisse über den realen Computer helfen, das Programm besser, effizienter zu gestalten. Dies ist ein wichtiger Gesichtspunkt, denn Effizienz wird in der Informatik gross geschrieben. Sprache und Computer sind die Werkzeuge des Informatikers. Für alle Ingenieursparten aber gilt seit jeher, dass ein Ingenieur in erster Linie seine Werkzeuge bestens kennen muss. Was gab es 1968 über Computer grundsätzliches zu lehren, und wie hat sich die Technik inzwischen verändert?

4. Des Computers Grundstruktur

Es war schon damals klar, dass neben der Fähigkeit zu programmieren auch Grundkenntnisse über Computer und deren Strukturen und Funktionsweise vermittelt werden mussten. Dazu ist es allerdings nötig, dass Studenten direkt mit einem Computer in Kontakt kommen können. Gerade dies jedoch war damals aber unmöglich, weil der vorhandene Grossrechner im Rechenzentrum versteckt und unzugänglich war. Programme und Daten wurden am Schalter in Form eines Lochkartenstapels abgegeben. Die Resultate erschienen am folgenden Tag auf einem Endlospapier als Zahlen und Grossbuchstaben. Von Interaktivität war keine Rede. Heute fragt man sich, wie unter diesen Umständen überhaupt jemand Lust am Arbeiten mit dem Computer finden konnte.

In Anbetracht dieser widrigen Umstände wurde für die genannten Unterrichtszwecke ein Kleinrechner angeschafft. Einigermassen erschwingliche Minicomputer waren erst einige Jahre zuvor auf dem Markt erschienen. Meine Wahl fiel auf den HP-2115 Rechner, dessen Aufbau sehr übersichtlich und zweckvoll war, was ich als unabdingbare Eigenschaft einstufte. Der Rechner und alle Funktionen waren gut erklärbar und verständlich. Sein Manual mit den Einzelheiten über seine Struktur und seine Programmierung umfasste nur 70 Seiten. Der Befehlssatz enthielt nur

16 Instruktionen, was in Anbetracht der damals üblichen Bewunderung für möglichst grosse Repertoires erstaunlich ist. Die technischen Daten lauten:

Wortlänge: 16 Bit
2 Register in der ALU
Speicher: 8192 Worte
Taktrate: 2 us
Tastatur, 24 Zeilen zu 80 Zeichen Display
Zeilendrucker, Lochstreifen (später Kartenleser)
Transistortechnik, Magnetkernspeicher

Die Programmierung fand mit Hilfe eines Assemblers statt. Programme waren also Folgen von Befehlen, wie z.B. addiere, subtrahiere, springe, etc. Der HP 2115 widerspiegelte direkt die grundlegende Rechnerstruktur nach J. von Neumann. Diese sieht bekanntlich zwei Einheiten vor: ein Steuerwerk und ein Rechenwerk, genannt ALU für *arithmetic logical unit*. Das Steuerwerk holt den jeweils nächsten Befehl aus dem Speicher in ein Instruktions-Register zur Interpretation und bestimmt die Adresse des folgenden Befehls (s. Abb. 1).

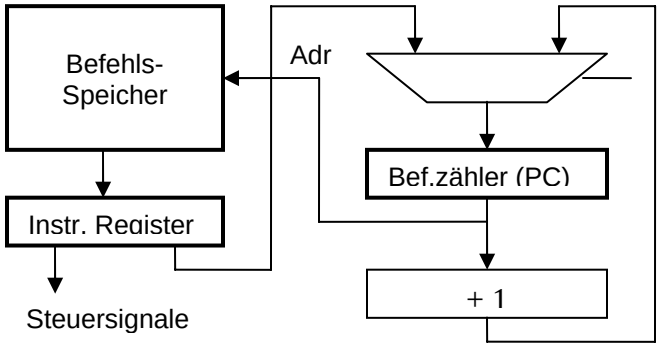

Abb.1. Steuerwerk und Befehlsspeicher

Das Rechenwerk besteht aus der arithmetisch-logischen Einheit und einem Register, damals *Akkumulator* genannt, weil darin Summen akkumuliert wurden (s. Abb. 2). Die wichtigste Errungenschaft nach von Neumann war die Möglichkeit, dass nun errechnete Resultate den Ablauf der Befehle beeinflussen konnten, indem der Multiplexer im Steuerwerk entweder die Adresse des nachfolgenden Befehls (PC+1) oder diejenige eines Sprungbefehls im Instruktionsregister auswählt. Die typischen Kriterien sind, ob ein Resultat gleich, kleiner, oder grösser als 0 ist.

Der Ablauf eines Programms besteht also aus der ständigen Wiederholung der zwei Phasen: Laden eines Befehls, und darauf Interpretation des Befehls und Bestimmung des nächsten Befehls.

Anmerkung: Hier sind Befehls- und Datenspeicher getrennt aufgeführt, was einer sog. Harvard-Architektur entspricht. Eine wichtige Grundidee von Neumann's war es, diese Speicher zu vereinigen. Die getrennte Anordnung hat neuerdings wieder einen Aufschwung erlebt, weil in manchen Anwendungen die Befehle in einem Festwertspeicher liegen, die beiden Speicher also auf verschiedenen Techniken basieren.

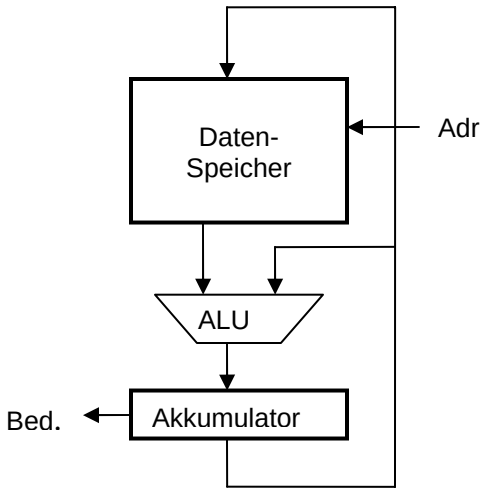

Abb. 2. Rechenwerk

Anmerkung: 1974 wurde im Institut der HP2115 durch einen DEC PDP-11 Rechner ersetzt. Dies war ein weitverbreiteter Multi-Register Kleinrechner mit 64K Byte Halbleiter-Speicher und Wechselplatte. Er wurde mit einem Grafik-Bildschirm (vector display) ergänzt. 1978 demonstrierte ich damit die erste Maus. Sie fand jedoch kaum Beachtung.

4. Erweiterungen und neue Konzepte

Diese bestechend einfache Struktur ist im wesentlichen auch heute noch das Fundament der modernen Rechner. Sie wurde allerdings in den folgenden Jahren und Jahrzehneten durch verschiedene Vorkehrungen ergänzt, um höhere Rechenleistungen zu erzielen. Diese haben sich derart entwickelt, dass daneben die Grundstruktur oft kaum mehr erkennbar bleibt. Es waren im Wesentlichen die folgenden Konzepte, welche die von Neumann'sche Architektur ergänzen sollten:

- Unterprogramm (Subroutine)
- Register-Satz anstelle des Akkumulators
- Index-Register, Adressberechnung
- Unterbrechbarkeit (Interrupt)
- Virtueller Speicher
- Speicherschutz und Betriebsmodi
- komplexe Befehle, reichhaltige Befehlssätze
- Pipelines und Caches

Wir wollen nun diese hinzugefügten Konzepte kurz erläutern und rechtfertigen.

Das Konzept des *Unterprogramms* wurde von Wheeler erfunden und ist seither in jedem Computer fest verankert. Technisch bedeutet es einfach, dass der Programmzähler abgespeichert und wieder eingesetzt werden kann.

Schon bald war klar geworden, dass Geschwindigkeit gewonnen und Programmlängen verkürzt werden können, wenn anstatt einem einzigen Akkumulator mehere Register vorgesehen werden. (So enthält z.B. der HP-2115 deren zwei). Es entstand daraus die *Multi-Register-Architektur*.

Als störend wurde sodann empfunden, dass Datenadressen fest in den Befehlen verankert sind. Um in Wiederholungen (Programmschleifen) jedes Mal andere Daten anzufordern (man denke an die Summierung von Elementen eines Arrays), mussten Befehle (zur Laufzeit) modifiziert werden. Dies verlangsamt nicht nur den Ablauf, sondern ist eine häufige Quelle von Programmierfehlern. Die Forderung nach invariantem Code wurde dadurch realisiert, dass ein oder mehrere Register

eingeführt wurden, deren Wert zur Laufzeit zu Datenadressen addiert wird. Diese Register wurden *Index-Register* genannt, weil ihr Wert meistens dem Index eines Arrayelements entspricht.. Etwas später wurden sie mit den Registern des Rechenwerks vereinigt (z. B. 1964 im IBM System 360).

Grossrechner verwendeten Magnetbänder und Magnetplattenspeicher. Diese zeichnen sich durch schnelle Datenströme aus; die Daten werden nicht als einzelne Zahlen, sondern in ganzen Blöcken übermittelt. Um Wartezeiten zu verringern, die Kopplung von Speichern und Rechner zu lockern, bedient man sich der Datenpufferung, wozu eigentlich ein eigener Prozessor nötig wäre, der jederzeit synchron mit dem Speicher arbeitet. Weil dieser Aufwand aber kostspielig ist, kam man auf die Idee, mehrere *logische Prozesse* durch den gleichen *physischen Prozessor* abwickeln zu lassen. Dies wurde ermöglicht durch eine Vorrichtung, die diesen Prozessor aufgrund eines externen Signals, ausgelöst z. B. durch den Plattenspeicher, von einem Prozess zu einem andern ablenkt, z. B. zu demjenigen, der Daten vom Plattenspeicher empfängt und in den Puffer ablegt. Dieser Vorgang wird *Interrupt* (Unterbrechung) genannt, und ein logischer Prozess wird heute als *thread* bezeichnet.

Damit war die Idee geboren, mehrere logische Prozesse durch einen einzigen, physischen Prozessor ablaufen zu lassen. Die Umschaltung des Prozessors von einem auf einen andern Prozess liess sich durch Interrupts bewerkstelligen, die von einem fixen Taktgeber in kurzen Zeitintervallen ausgelöst wurden (z. B. alle 20ms). Dies war das Konzept der *Time-sharing-Systeme*, wo eine Anzahl von Benutzern quasi gleichzeitig von einem einzigen Rechner bedient wurden und so den Eindruck suggeriert erhielten, einen ganzen Rechner für sich allein zu besitzen. Sogar das Gefühl der Interaktivität war damit zurückgekehrt.

Durch die Erfindung von Mehrbenutzer- und Time-sharing-Systemen wurde es nötig, Vorkehrungen zur zentralen Speicherverwaltung und zum Schutz von Programmen gegen Fehler in anderen, gleichzeitig ablaufenden Programmen zu treffen. In diesem Zug wurden Konzepte wie virtueller Speicher, Speicherschutz und Betriebsmodi erfunden. Es soll an dieser Stelle nicht weiter auf sie eingegangen werden, sondern wir stellen lediglich fest, dass sie enorm zur weiteren Verkomplizierung der Rechner beigetragen haben.

Auffällig war der Trend zu immer *reichhaltigeren Befehlssätzen*. Damit sollte vor allem die Programmierung gewisser Anwendungen erleichtert werden. Man denke an Befehle zum Suchen, Vergleichen und Kopieren von Zeichenfolgen (strings). Gleichzeitig wurde mit den komplexen Befehlen die Verkürzung des Codes angestrebt. Auch sollte mit geeigneten Befehlssätzen und Strukturen die Kluft zwischen Code und Programmiersprache verkleinert werden. Mit dem Burroughs B5000 Computer und der Sprache Algol wurde diesen Bemühungen ein würdiges Denkmal gesetzt. Die Dichte des Befehls-Codes spielte damals wegen der beschränkten Speichergrösse eine wichtige Rolle. Jedenfalls entstanden unglaublich komplizierte Rechner mit riesigen Befehlssätzen und einer grossen Auswahl von Adressierungsmodi. Gewisse Instruktionen waren elementar und schnell, andere jedoch kompliziert und langsam, und stellten gleichsam Unterprogramme dar.

5. RISC

In diese Umgebung hinein platzte Mitte der 80er Jahre die Antithese, der RISC (Reduced Instruction Set Computer). “Schluss mit den komplizierten Instruktionen und zurück zur einfachen Architektur mit nur wenigen, elementaren Befehlen”, lautete die Devise. Jede Instruktion sollte nur einen einzigen Takt in Anspruch nehmen, was eine Erhöhung der Taaktrate zuliess. Inzwischen hatte sich auch die Erkenntnis durchgesetzt, dass in Zukunft Computer nicht mehr “von Hand” programmiert würden, sondern mittels Programmiersprachen und Compiler, was die Wünschbarkeit spezieller und komplexer Instruktionen deutlich abschwächte..

Die Konsequenz war die Verlagerung der Komplexität aus dem Computer in den Compiler, aus der Hardware in die Software. Doch wurde dies getrost in Kauf genommen, denn Computer waren inzwischen derart schnell geworden, dass Compilationszeiten keine relevate Rolle mehr spielten. Mit dem RISC war eine neue Computer-Generation geboren worden. Computer waren wieder überschaubar und erklärbar geworden.

Hingegen ermöglichte diese Aenderung die Einführung von weiteren, effizienzsteigernden Massnahmen. Man hatte nämlich bemerkt, dass vor allem Instruktionen, aber auch Daten meistens sequetiell, der Reihe nach, gelesen werden. Da Prozessoren in wesentlich höherem Masse schneller geworden waren als Speicher, konnte Effizienz am ehesten dort gesteigert werden, wo ein Flaschenhals lag, eben beim Speicher. Die Lösung stellten *Pipelines* und *Caches* dar, Mechanismen zur Entkopplung von Rechner und Speicher. Allerdings machten sie die durch die RISC Struktur erzielten Vereinfachungen teilweise wieder rückgängig. Das Resultat war, dass sowohl Computer als auch Compiler, also Hardware *und* Software komplexer wurden, deren Schnittstzelle jedoch einfacher. Endziel aber blieb die Steigerung der Gesamtleistung.

Dieser Trend hält noch heute an. Zwar sind auch Speicher schneller geworden. Ein Grund dafür ist, dass nicht mehr einzelne Zahlen und Bytes ausgelesen werden, sondern stets ganze Blöcke. Dies passt natürlich in das Schema mit Zwischenspeichern (caches, s. Abb. 3)). Es wird heute aber sogar erwogen, noch einen weiteren Schritt in der Komplexitätsverlagerung hin zur Software zu tun, indem auch die Verwaltung der Caches explizit im Code enthalten ist, und damit bereits vom Compiler vorweggenommen wird.

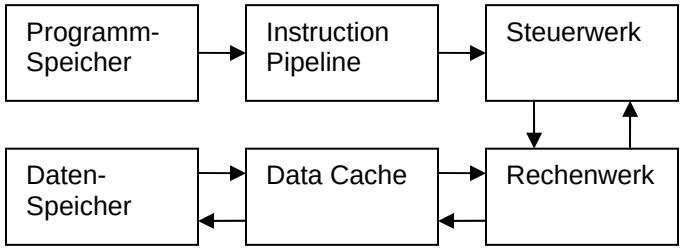

Abb. 3. Pipeline und Cache

Vor allem aber haben Speicher an Kapazität zugelegt. Während noch 1980 in Kilobytes gerechnet wurde, spricht man heute von Gigabytes (Faktor 10^6). Dies bedeutete, dass das Gütekriterium Code-Dichte nun keine Rolle mehr spielt. Es ist in der Tat erstaunlich, wie stark das Medium der Implementation bestimmt, welche Struktur die geeignetste ist. Die vielgepriesene Abstraktion erfährt damit offensichtlich ihre Grenzen.

Positiv zu vermerken ist, dass sich die Einsicht langsam durchsetzt, dass der gegenseitige Schutz gleichzeitig residenter Programme (memory protection) bei der Programmierung mit Sprachen anstatt “von Hand” jetzt vom Compiler übernommen werden kann (Typenprüfung), und dass damit eine signifikante Quelle von Komplexität der Hardware entfällt, resp. entfallen könnte.

Nebenbei sei hier noch auf eine Ausnahmeerscheinung hingewiesen, die bereits 1963 alle Charakteristiken des RISC vorweggenommen hatte, die CDC-6000 Architektur von Seymour Cray, und zwar ungefähr gleichzeitg mit seinem ideellen Gegenteil, dem B-5000. Daher wird das Akronym RISC heute gelegentlich auch als Really Invented by Seymour Cray interpretiert.

6. Multi-core Chips

Die jüngste Effizienz-steigernde Neuerung ist die Anordnung mehrerer Prozessoren auf demselben Bauteil (Chip). Weil man an physikalische Grenzen stiess - zwar nicht bei der weiteren Miniaturisierung, jedoch bei der Erhöhung der Taktrate - schien der stetigen Leistungssteigerung ein Ende bevor zu stehen. Den Ausweg fanden die Chip-Hersteller darin, mehrere Prozessoren (cores) auf dem gleichen Chip zu platzieren. Die Software-Entwickler werden damit zwangsläufig mit dem schwierigen Problem konfroniert, mehrere Prozesse zu unterhalten und zu koordinieren. Dies jedenfalls, wenn sie die neuen Chips optimal ausnützen wollen.

Zwar sind die Herausforderungen der Multiprogrammierung keine unbekannten Geister. Man hat sie aber in der Vergangenheit möglichst ruhen lassen und nur bemüht, wenn dies unvermeidbar war, wie z.B. bei eingebetteten Sytemen zur Daten-Aquisition und zur Echtzeit-Steuerung von

Maschinen. Nun jedoch wird die Multiprogrammierung unumgänglich, zusammen mit der generellen Frage, wie ein Prozess durch mehrere, gleichzeitige, sequentielle Abläufe ersetzt werden kann. Es bedarf keiner prophetischen Fähigkeiten um vorauszusehen, dass die Software noch weiter an Komplexität zunehmen wird. Der Trend, die Hardware schnell und daher einfacher zu machen, und alles weitere der immer komplizierteren Software anzulasten, dürfte sich damit noch verstärken.

Doch schon heute sind Computer kaum mehr erklärbar!

7. Was ist aus der Geschichte zu lernen?

Bei jeder Schaffung eines Artifakts ist es nötig, verschiedene Varianten gegen-einander abzuwägen, deren Vor- und Nachteile zu erkennen, und Kompromisse zu schliessen. Jeder Ingenieur weiss dies. Dabei ist es wichtig, das Gesamte im Auge zu behalten. Im vorliegenden Fall der System-Entwicklung schliesst dies Hardware *und* Software mit ein. Vereinfachungen auf der einen ergeben meistens Verkomplizierungen auf der andern Seite, Verkomplizierungen auf der einen aber nicht unbedingt Vereinfachungen auf der andern. Um deren Summe zu minimieren, ist es nötig, Kenntnisse auf beiden Gebieten vorzuweisen.

Zwangsläufig schliessen wir daraus, dass der angehende Informatik-Ingenieur nicht nur die traditionelle, obligate Ausbildung auf dem Gebiet der Software und Programmierung erhalten soll, sondern auch *solide Grundkenntnisse in Hardware*, in Computerstrukturen.

Diese Forderung ergibt sich auch aus der Tatsache, dass es heute programmierbare Bauteile (programmable devices) gibt. Diese erlauben die Konstruktion komplexer Schaltungen, selbst ganzer Prozessoren, rein auf der konzeptionellen Ebene, d.h. ohne Aufbau mit konkreten, materiellen Schaltungskomponenten. Die Entwicklung einer Schaltung wird damit zum “Software-Prozess”; das Mittel dazu ist allein ein Beschreibungs-Formalismus und sein Compiler. Die zwei hauptsächlichen Klassen sind die *Programmed Logic Devices* (PLD) und die *Field Programmable Gate Arrays* (FPGA). Wenn sie auch effizienzmässig nicht mit Custom Design konkurrenzfähig sind, so kommt ihnen schon heute eine enorme Rolle in der Entwicklung von Prototypen von Schaltungen zu. Sie sind das ideale *Experimentierfeld,* das stets ohne jeglichen Materialverschleiss auskommt.

Ermöglicht wird diese neue Art von Flexibilität durch eine grosse Menge elementarer Schaltungskomponenten. Die für einen Design relevanten werden individuell ausgewählt – nicht durch Verdrahtung, sondern durch Schalter, deren Stellung in einem eingebauten Festwertspeicher (configuration memory) bestimmt wird (s. Abb. 4). Die nicht relevanten – meistens deren Mehrzahl - liegen brach.

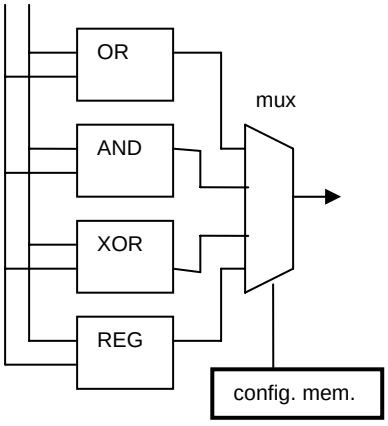

Abb. 4. Prinzip des FPGA

Faszinierend ist die Feststellung, dass neurdings Hardware ganz ähnlich wie Software beschrieben wird. In beiden Fällen dienen zur Definition von Schaltungen, resp. von Programmen

formale Sprachen. Sie sind sich sogar verblüffend ähnlich. Wer hingegen glaubt, in den beiden Sparten die gleichen Denkmodelle anwenden zu können, der siehst sich früher oder später getäuscht. Schaltungen sind statische Gebilde. Alles läuft gleichzeitig ab, “parallel”, wie der Programmierer es nennt. Dieser jedoch ist gewohnt, mit sequentiellen Abläufen umzugehen. Diesen Unterschied zu begreifen, gehört meines Erachtens zur Grundausbildung eines jeden modernen Informatikers, jedenfalls eines jeden zukünftigen Informatik- oder Elektroingenieurs mit Hochschulabschluss.

Wir haben gezeigt, dass der Trend zur Komplexität stetig fortschreitet und leider irreversibel ist. Was die Entropie der Physik, ist der Informatik die Komplexität! Die neuen Produkte der Technik zu verstehen wird immer schwieriger. Die Zahl derjenigen, die sie nicht verstehen, gleichzeitig sich aber von ihnen abhängig gemacht hat, wächst. Dieser Vorgang hat leider auch soziale Folgen. Wer die Welt noch versteht, hat Vorteile. Die Kluft zwischen den Experten und den Laien verbreitert sich zum Nachteil der Gesellschaft rasch und nachhaltig.

Abhilfe kann hier nur eine breiter angelegte Bildung (nicht nur Ausbildung) verschaffen. Es kann doch nicht angehen, dass jedermann tagtäglich Geräte benutzt, sie stets bei sich trägt, von ihnen unzertrennlich ist, von denen er längst abhängig ist, von deren Grundlagen er jedoch nicht die allergeringst Ahnung besitzt, er nicht im Entferntesten weiss, was alles an menschlicher Intelligenz und Können in sie eingeflossen ist.

Aber so ist es heute! Es ist das Resultat unserer Bildungs-Philosophie. Naturwissenschaften spielen in den Lehrplänen eine kümmerliche Rolle, und von Technik ist keine Rede. Aber Naturwissenschaften und Technik prägen heute unsere Welt. Naturwissenschafter und Techniker sind die neuen Kulturschaffenden. Vielleicht ist ein zweites 1968 nötig, um ein Aufwachen einzuleiten.

Dass diese Fächer eine prominente Rolle in unserem Schulwesen spielen sollen, ist auch das Anliegen, das unser Gewerbe und die Industrie mit dem Jahr der Informatik 2008 zum Ausdruck bringen. Ihm liegt der Wunsch zugrunde, dass unser Land auch in Zukunft in der Welt einen Platz behalten und seinen Wohlstand wahren kann. Ihm liegt auch die Erkenntnis zugrunde, dass dies nicht ohne Anstrengung, ohne Leistungen geschehen kann. Und hier sind wir wiederum beim famosen Jahr 1968 gelandet, das gerade diese Tatsache verdrängen wollte. Und es ist ihm teilweise gelungen. Der verbreitete Drang nach viel Konsum und nach “instant satisfaction” anstelle von eigenem Einsatz und Beitrag macht dies sichtbar.

Vielleicht liegt diese Einstellung auch hinter der vielbeklagten Abneigung gegen naturwissenschaftliche und technische Schulfächer. Es sind dies Fächer, die Exaktheit verlangen, die unklare Antworten nicht vergeben, harte Fächer eben. Gerade diese aber sind verpönt, denn sie erfordern Leistung in Form von exaktem Denken. Dies aber verlangt Konzentration, Ausdauer, Training. Wir aber “joggen” viel lieber mit den Beinen als mit dem Kopf.

Man missverstehe mich nicht. Es wird hier nicht verlangt, dass jedermann lernen sollte, wie man Computer und Mobiltelefone baut. Aber gewisse Kenntnisse über deren Grundlagen, Prinzipien und Grenzen wären angemessen. Sie gehören doch heute zur Allgemeinbildung. Wo sie fehlen, bürgern sich vielzitierte Fehlbegriffe ein, wie “den kleinsten gemeinsamen Nenner finden” (er ist immer 1, und muss daher nie gesucht werden), oder “erneuerbare Energie” (Energie bleibt stets erhalten), oder “Atomstrom” (Strom heisst Fluss von Elektronen). Diese Schlagwörter sind untrügliche Symptome einer Fehlleistung unseres Bildungswesens.

Aus all diesen Schlussfolgerungen geht klar hervor, dass Schulen, Universitäten, und insbesondere Informatikabteilungen heute vor wichtigen Aufgaben stehen. Möge es ihnen gelingen, diese wahrzunehmen.